Web NDL Authorities SPARQL API Specification

National Diet Library of Japan
March 31th, 2014

Contents

1 The Outline of the Web NDLA SPARQL API

The Web NDL Authorities (Web NDLA) stores the name authority information as RDF (Resource Description Framework) data. RDF data can be queried by SPARQL (SPARQL RDF Query Language), and the Web NDLA has the function to respond the SPARQL query.

This document describes the RDF data structure of the Web NDLA, and explains how to query it with SPARQL.

1.1 SPARQL query API

1.1.1 Basic SPARQL Query

A user can search any authority data in Web NDLA with **SPARQL Query**. The next example is a query to find the authority URI of the subject heading "図書館" (library).

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj rdfs:label "図書館"
}
```

The search result will be returned by sending this query to NDLA though API using the following procedure.

1.1.2 Request URI and Parameters

A SPARQL query to the Web NDLA is to be requested against the following URI (endpoint).

```
http://id.ndl.go.jp/auth/ndla
```

There are two parameters as shown in Table1.

Table 1: Web NDLA API Parameters

parameter	value
query	URL encoded SPARQL query string
output	result format (`xml | json | turtle*`) *turtle is for `DESCRIBE`, `CONSTRUCT` only

A request to get results as XML is of the following form.

```
http://id.ndl.go.jp/auth/ndla?query={URL encoded query}&output=xml
```

With the previous query being URL encoded, the entire request will be the following.

```
http://id.ndl.go.jp/auth/ndla?query=PREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F
2000%2F01%2Frdf-schema%23%3E%0D%0ASELECT+*+WHERE+%7B%0D%0A%09%3Fsubj+rdfs%3Alabel
```

```
+%22%E5%9B%B3%E6%9B%B8%E9%A4%A8%22%0D%0A%7D%0D%0A&output=xml
```

There are four SPARQL query forms: `SELECT` to find values, `ASK` to see if the matching data exists, `CONSTRUCT` to create new RDF graphs with the matching value, and `DESCRIBE` to obtain an explanation graph for resources.

For `DESCRIBE` or `CONSTRUCT` queries, value `turtle` can be specified for the result format parameter (output).

SPARQL query will be discussed in chapter 2, and four query forms will be explained in 2.8.

1.1.3 Result Format

As the result of a request, a list of bound variables (for `SELECT`), an RDF graph (for `DESCRIBE` and `CONSTRUCT`) or a truth value (for `ASK`) will be returned, according to the type of the query.

The format of the binding list and the truth value is varied by the `output` parameter as shown in Table 2.

Table 2: Web NDLA API output Parameters

output parameter	format
xml	XML as specified in SPARQL Query Results XML Format
json	JSON as specified in SPARQL 1.1 Query Results JSON Format

The resulting XML for the previous query (`output=xml`) will be:

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="subj"/>
  </head>
  <results>
    <result>
      <binding name="subj">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00573385</uri>
      </binding>
    </result>
  </results>
</sparql>
```

The detail of XML format and JSON format will be shown in 3.1 and 3.2 respectively.

The result format of RDF graph is shown in Table 3.

1.2 Authority Record RDF Graph and SPARQL

Authority records in the Web NDLA are expressed as RDF graphs. A SPARQL query is constructed with "patterns to find in a graph". This section explains the outline of how to write patterns in an RDF graph as SPARQL query.

Table 3: Web NDLA API RDF Graph Formats

output parameter	Format
xml	RDF/XML format
json	RDF/JSON format
turtle	Turtle format

1.2.1 The RDF Graph of the Authority Record

RDF describes a basic information in a way that a thing (e.g. an authority record = subject) has a property or characteristic (predicate) which has a value (object). This subject - predicate - object relation is called an **RDF Triple**. Each element of a triple is named (identified) by a URI[1]. An object can be **literal** instead of a URI.

A set of RDF triples is called an **RDF Graph**. In an RDF graph, triples that share the same URI is connected, resulting a large network of information (Figure 1).

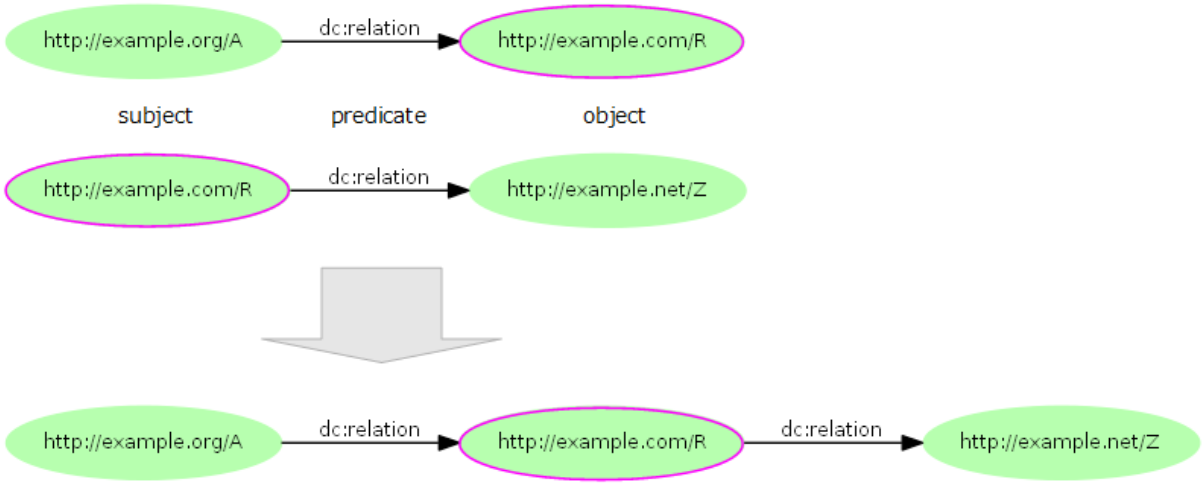

Figure 1: Triples are connected via a common URI.

Authority records in the Web NDLA are represented by RDF. Each authority record is identified by a URI, and the relationship between records (e.g. broader or alternative term) is described as RDF triple. Thus authority records are connected, and form a large RDF graph (Figure 2).

Moreover, because the Web NDLA references external authorities such as VIAF or LC authorities, the RDF graph of the Web NDLA goes beyond the National Diet Library of Japan, and becomes a part of LOD (Linked Open Data) cloud (The whole model of an authority record will be discussed in chapter 4).

[1]Actually, it is IRI (Internationalized Resource Identifier, the extension of URI so that non ASCII characters can be appear in an identifier string) that is used for naming. This document uses more familiar URI as the term of identifier, but it should be read as IRI.

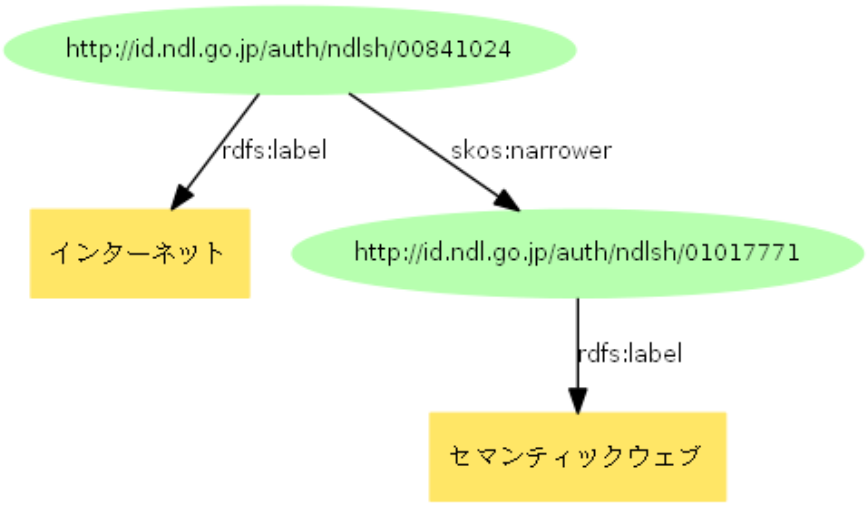

Figure 2: The subject URI of "セマンティックウェブ" (Semantic Web, http://…01017771) is also the object of skos:narrower property of "インターネット"(Internet, http://…00841024) , hence these triples are connected.

1.2.2 Partial Graph Pattern and Search

With SPARQL, a user will write a partial pattern of an RDF graph that contains unknown data (variables), find values that match this pattern from the target graph, and retrieve the result as sets of values.

For example, in order to find authorities whose label (`rdfs:label`) are the narrower terms (`skos:narrower`) of "インターネット", the partial graph pattern will be as Figure 3.

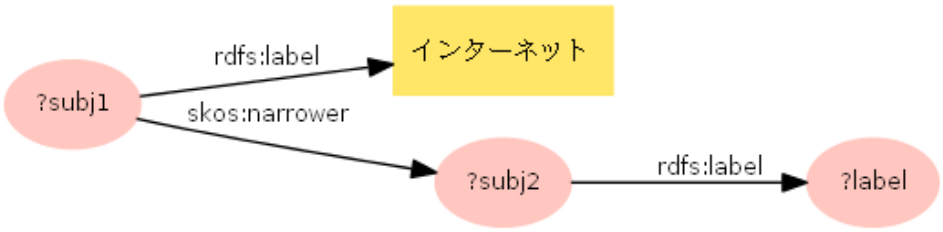

Figure 3: A partial graph pattern where the variables 'subj2' and 'label' represent the narrower term and its label.

The search will find the matching sub graphs from the Web NDLA, and returns values of the corresponding variables as in Table 4.

1.2.3 Writing a SPARQL Query

In a SPARQL query, partial graph patterns are expressed with Turtle-like syntax. Unknown elements are written as variables, whose name begin with a letter `'?'`. The partial graph pattern of Fig. 3

Table 4: The results retrieved by the partial graph pattern

subj2	label
http://id.ndl.go.jp/auth/ndlsh/01017771	セマンティックウェブ
http://id.ndl.go.jp/auth/ndlsh/00969901	バーチャルプライベートネットワーク
http://id.ndl.go.jp/auth/ndlsh/00865280	イントラネット

will be written as follows.

```
?subj1 rdfs:label "インターネット" ;    # Internet
    skos:narrower ?subj2 .
?subj2 rdfs:label ?label .
```

The steps to search the graph with this pattern are:

1. If graph patterns use prefixed names, map those prefixes to URI by keyword `PREFIX`[2],
2. put keyword `SELECT` followed by space separated variables to retrieve (similar to columns in SQL),
3. then put keyword `WHERE` followed by patterns enclosed by `{}`.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?subj2 ?label
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

In this example, two variables `?subj2 ?label` are listed after `SELECT` in order to get the values shown in Table 4. By writing `*` instead of variables list, all values of the variables used in the graph pattern will be retrieved[3].

```
SELECT * WHERE {
 ...
```

For `ASK`, `CONSTRUCT` and `DESCRIBE` queries, `SELECT` clause is replaced by each corresponding construct while `PREFIX` clause and `WHERE` clause parts are the same (See 2.8 for detail).

[2]Although some of following examples may omit PREFIX clause for simplicity, all prefixes must be mapped to URIs by PREFIX.

[3]In these examples, first one has a new line before WHERE, while the second one does not. Since a new line character is regarded as a white space in SPARQL syntax, it does not make any difference.

2 SPARQL RDF Query Language

The Web NDLA uses ARC2 library[4] which supports SPARQL Query Language for RDF 1.0[5]. This chapter explains SPARQL query that can be used with the Web NDLA.

2.1 Basic Syntax

Most forms of SPARQL query contain a set of triple patterns called a **basic graph pattern**. These patterns are expressed by Turtle-like syntax.

2.1.1 Triple Patterns

A **triple pattern** is like RDF triple except that each of the subject, predicate and object may be a variable. A triple pattern consists of the following components:

- A term enclosed by `<>` is a **URI**[6]. It can be an absolute URI, or a relative URI combined with `BASE` clause.
- A URI can be expressed as a **prefixed name** in the form of `prefix:localname`. The prefix label must be associated with a URI by `PREFIX` clause.
- A value enclosed by `""` or `''` is a **literal**, with either an optional language tag (introduced by `@`) or an optional datatype IRI / prefixed name (introduced by `^^`).
- If a prefixed style name starts with `_:`, it is a **blank node** in a graph. A blank node can also be expressed by `[]` form as in Turtle[7].
- A variable is an alphanumeric string[8] prefixed by either `?` or `$`. A variable name may contain an under bar (`_`), but not a hyphen (`-`). Query variables in SPARQL queries have global scope.

A special single character '`a`' may be used as an abbreviation of predicate `rdf:type` which relates a subject to a class. A user may find it convenient because `PREFIX` clause for `rdf:` is not necessary if there is no other term from that namespace.

The next is an example of abbreviation '`a`' which relates the subject (variable) to `foaf:Person` class. This query will find person entity resources (see 4.1) in the authority records.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
    ?who a foaf:Person
}
```

Without using abbreviation '`a`', the above query will be as follows.

[4] https://github.com/semsol/arc2

[5] http://www.w3.org/TR/rdf-sparql-query/

[6] It is actually an IRI in SPARQL, too. This document uses more familiar term URI in place of IRI.

[7] Blank nodes in graph patterns act as non-distinguished variables, not as references to specific blank nodes in the data. Therefore, they will match even URIs or literals in RDF graph, though those values cannot be retrieved.

[8] The Web NDLA does not support non alphanumeric variable names, although SPARQL specification permits them.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
    ?who rdf:type foaf:Person
}
```

A character `#` is a comment marker if presents outside literals and URIs. After the maker up to the end of line will be a comment.

New line and TAB characters are treated as white spaces.

2.1.2 Datatype and Language Tag in Literal

A literal may be a plain literal, or may have a datatype (e.g. date, integer etc.) or a language tag. A care should be taken because a plain literal and a typed literal or a language tagged literal are treated as different values even if their lexical forms (string parts) are the same.

- The Web NDLA does not use datatypes. For example, a record creation date is always in the form of `"2013-12-15"` in the Web NDLA, while it might be expressed as `"2013-12-15"^^xsd:date` in some other datasets.
- The Web NDLA uses language tags only for the transcriptions of structured labels. For example, the subject heading "図書館" (library) has two transcriptions `"トショカン"@ja-Kana` and `"Toshokan"@ja-Latn`, so that users can distinguish Japanese-kana and Japanese-romaji transcriptions.

2.1.3 Graph Patterns

A **graph pattern** is a set of triple patterns. Each triple pattern is delimited by a period (`.`)[9].

There are abbreviated ways of writing some common triple pattern constructs, same as in Turtle: The common subject can be abbreviated by `;` so that the rest are written as predicate-object list. The common predicate is abbreviated by `,` and the rest are just object list.

```
?subj rdfs:label ?label ;
    skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> ,
        <http://id.ndl.go.jp/class/ndc9/694.5> .
```

The above example contains three triple patterns which share the common subject `?subj`. Also, the common predicate `skos:relatedMatch` is omitted in the last line with `,` at the end of previous line. Without shortcut, those triple patterns will be written as follows.

```
?subj rdfs:label ?label .
?subj skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> .
?subj skos:relatedMatch <http://id.ndl.go.jp/class/ndc9/694.5> .
```

[9] A period is just a delimiter, and not required at the end of a triple pattern (different from Turtle), hence it is not necessary to place a period at the end of a graph pattern.

2.1.4 Filters

SPARQL provides `FILTER` to test or restrict the values. For example, the following query matches records whose labels contain "夏目" (`FILTER` will be further explained in the section 2.6).

```
?uri rdfs:label ?label .
FILTER regex(?label, "夏目")            # Natsume
```

2.2 Groups of Graph Patterns

SPARQL graph pattern matching is defined in terms of combining the results from matching **basic graph patterns**.

A sequence of triple patterns (with optional filters) comprises a single basic graph pattern. Any other graph pattern terminates the basic graph pattern.

A set of one or more graph patterns delimited by `{}` is called **group graph pattern**. The `WHERE` clause of a query consists of the keyword followed by one group graph pattern.

Group graph patterns can be nested.

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;    # Internet
        skos:narrower ?subj2 .
    {?subj2 rdfs:label ?label }
}
```

In the above example, the `WHERE` clause has one group graph pattern, within which there are one basic graph pattern and one group graph pattern. The inner group graph pattern consists of one basic graph pattern.

2.2.1 Graph Patterns and Filters

A `FILTER` restricts solutions over the whole group graph pattern in which the filter appears. Within the same group graph pattern, the filter has the same scope regardless its position.

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    FILTER regex(?label, "ネット")          # net
}
```

The above example has the same solutions as bellow.

```
WHERE {
    FILTER regex(?label, "ネット")
```

```
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

2.2.2 Graph Patterns and Blank Node ID

Labels for blank nodes (blank node IDs) are scoped to the basic graph pattern. A label can be used in only a single basic graph pattern in any query[10]. Therefore, the following example is valid:

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower _:s2 .
    _:s2 rdfs:label ?label .
}
```

while the next one is an error.

```
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower _:s2 .
    {_:s2 rdfs:label ?label }
}
```

In the second example, the inner `{}` divides the query into two basic graph patterns, which cannot share the same blank node id (`_:s2` in this case).

2.3 Optional Pattern Matching

In a query with basic graph patterns, all variables in the query must have matches to have a solution. For example, in the next query, narrower terms of "インターネット" will not be retrieved unless they have their related terms.

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label ;
        skos:related ?subj3 .
    ?subj3 rdfs:label ?rels .
}
```

[10] Although blank nodes act like variables, their scope is different from that of variable which is global.

In order to make non-required variables optional, enclose the partial graph that contain those variables as a group graph pattern, then concatenate it to the required pattern by `OPTIONAL` keyword.

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
    }
}
```

The above query will find all narrower terms of "インターネット", as well as the related terms of them if they have any.

2.3.1 Multiple Optional Patterns

A query can have multiple `OPTIONAL` patterns.

```
pattern 1 OPTIONAL {pattern 2} OPTIONAL {pattern 3}
```

Those patterns are left-associative. The above one is the same as the next:

```
{pattern 1 OPTIONAL {pattern 2}} OPTIONAL {pattern 3}
```

2.3.2 OPTIONAL and FILTER

A `FILTER` can be applied to an optional pattern. The next example is a query that will find the narrower terms of "インターネット" and also find their related terms which contain "情報" (information).

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
        FILTER regex(?rels, "情報")
    }
}
```

Note that the scope of a FILTER is the group graph pattern. If the FILTER keyword is placed outside the OPTIONAL pattern as in the next example, it will affect the entire WHERE clause.

```
SELECT ?subj2 ?label ?subj3 ?rels
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
    OPTIONAL {
        ?subj2 skos:related ?subj3 .
        ?subj3 rdfs:label ?rels .
    }
    FILTER regex(?rels, "情報")
}
```

Because this FILTER examines whether ?rels contains "情報" for all solutions, any solution where the narrower term does not have related term (no binding for ?rels) will be excluded from the results set.

2.4 UNION for Matching Alternatives

To find results that match any of alternative patterns, join group graph patterns by UNION keyword.

For example, the next query will find subject headings that have classification code ND633 from National Diet Library Classification (NDLC) or 547.483 from Japan Decimal Classification 9th edition (NDC9).

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
prefix xl: <http://www.w3.org/2008/05/skos-xl#>
prefix ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT *
WHERE {
    {
        ?subj
            skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/ND633> ;
            rdfs:label ?dcndl .
    } UNION {
        ?subj
            skos:relatedMatch <http://id.ndl.go.jp/class/ndc9/547.483> ;
            rdfs:label ?ndc9 .
    }
}
```

2.5 RDF Dataset and Graph

A SPARQL query is executed against an **RDF Dataset** which represents a collection of graphs. An RDF Dataset comprises one **default graph**, which does not have a name, and zero or more **named graphs**, each of which is identified by an URI.

A query involves any of graphs in a dataset. The graph that is used for matching a basic graph pattern is the **active graph**. `GRAPH` keyword (see 2.5.1) specifies the active graph. If it does not present, the default graph is used as the active graph.

The dataset in the Web NDLA consists of two named graphs shown in Table 5.

Table 5: Graphs in Web NDLA dataset

data type	graph URI
Subject Headings	http://id.ndl.go.jp/auth/ndlsh
Name Authorities	http://id.ndl.go.jp/auth/ndlna

Also, the merge of those two graphs is the default graph in the Web NDLA.

2.5.1 GRAPH Keyword

In a `WHERE` clause, a `GRAPH` followed by a graph URI will set the active graph. After the URI, place the group graph pattern to search against this active graph.

```
SELECT * WHERE {
    GRAPH <http://id.ndl.go.jp/auth/ndlna> {
        ?s rdfs:label "インターネット"
    }
}
```

The above example will find "インターネット" in the Name Authority graph ("インターネット" as a corporate name will match).

If a variable follows `GRAPH` keyword, the result will have the graph URI where the group graph pattern has match.

```
SELECT * WHERE {
    GRAPH ?g {
        ?s rdfs:label "インターネット"
    }
}
```

The above query returns the result shown in Table 6, which tells that each of Subject Heading and Name Authority has a record "インターネット".

With multiple `GRAPH` keywords, a query will be executed against each graph. The same variable can be used across those graphs.

The next example will find the records that share the same labels in Subject Heading and Name Authority by using the same variable `?label` for both graphs.

Table 6: The search result of "インターネット" with graph name as variable

g	s
http://id.ndl.go.jp/auth/ndlsh	http://id.ndl.go.jp/auth/ndlsh/00841024
http://id.ndl.go.jp/auth/ndlna	http://id.ndl.go.jp/auth/ndlna/001144835

```
SELECT * WHERE {
    GRAPH <http://id.ndl.go.jp/auth/ndlsh> {
        ?sh rdfs:label ?label
    }
    GRAPH <http://id.ndl.go.jp/auth/ndlna> {
        ?na rdfs:label ?label
    }
}
```

2.5.2 FROM Clause

A clause `FROM <graph URI>` before `WHERE` clause indicates the graph to be used to form the default graph. `FROM NAMED <graph URI>` will introduce that graph as a named graph. Those keywords can be used multiple times (i.e. the query will be executed against multiple graphs).

In the Web NDLA, `FROM` clause acts to restrict the target graph[11]. The next example will find "インターネット" in the Name Authority, and have the same results as the query with `GRAPH` keyword.

```
SELECT *
FROM <http://id.ndl.go.jp/auth/ndlna>
WHERE {
    ?s rdfs:label "インターネット"
}
```

2.6 Restricting Values with FILTER

`FILTER` restricts solutions that match the graph pattern, by excluding solutions where any `FILTER` expression evaluates to FALSE.

```
?book ex:price ?price .
FILTER (?price < 2000)
```

In the above example, only solutions whose `?price` value is less than 2000 will be returned, others being excluded from the matching sets.

[11] Some services introduce external dataset by FROM, although the Web NDLA does not. FROM and FROM NAMED behave identically in the Web NDLA: specified graph will be incorporated to both default graph and named graph.

2.6.1 Comparison and Logical Operators

Like many programming languages, FILTER expression can have comparison and logical operators shown in Tables 7 and 8 respectively. An expression has to be enclosed by `()`.

Table 7: SPARQL Comparison Operators

Operator	TRUE condition
`A = B`	A is equal to B
`A != B`	A is not equal to B
`A > B`	A is greater than B
`A < B`	A is less than B
`A >= B`	A is greater than or equal to B
`A <= B`	A is less than or equal to B

Table 8: SPARQL Logical Operators

Operator	TRUE condition
`A || B`	Either A or B is TRUE (OR)
`A && B`	Both A and B are TRUE (AND)
`! A`	Not A (NOT)

Expressions are evaluated based on the data types of A and B. Suppose A is 10 and B is 5, the expression `A > B` evaluates to TRUE if those are numeric values, and to FALSE if string values. A and B must have the same types to be compared. In the Web NDLA, all literal values are strings (no data type), however, they are converted to numbers automatically if both A and B can be treated as numerical values[12].

For numerical values, arithmetic operators can be applied. The next example restricts the solutions to have 10 or more difference between max and min values.

```
?what ex:height ?max ;
    ex:low ?min .
FILTER (?max - 10 >= ?min)
```

2.6.2 Test Functions

Functions in Table 9 are provided to test values, e.g. whether they are literals, blank nodes etc.

The combination of `OPTIONAL` pattern and negation of `bound()` can be used to find solutions that do NOT have a value of particular property. The next example will find any living person (who has birth date but does not have death date).

[12] In general, use accessors such as STR() in 2.6.3 to align data types when operands have different types.

Table 9: SPARQL Test Functions

Function	TRUE condition
`bound(A)`	A is bound to a value
`isIRI(A)`	A is an IRI
`isURI(A)`	A is a URI
`isBLANK(A)`	A is a blank node
`isLITERAL(A)`	A is a literal

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
WHERE {
    ?who a foaf:Person; foaf:name ?name ;
        rda:dateOfBirth ?bdate .
    OPTIONAL {
        ?who rda:dateOfDeath ?ddate.
    }
    FILTER (!bound(?ddate))
}
```

Note that dateOfDeath has to be `OPTIONAL`. Otherwise, the graph pattern requires `?ddate` while `FILTER` excludes results that have matching `?ddate`, resulting empty solution set[13].

2.6.3 Accessors

Operators in Table 10 are accessors to obtain values other than truth value.

Table 10: SPARQL Accessors

Operator	Return Value
`str(A)`	the lexical form of A (simple string)
`lang(A)`	language tag value of A
`datatype(A)`	datatype URI of A

For example, with `lang()` accessor, a user can retrieve Romaji transcription of headings that match a pattern.

```
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT * WHERE {
```

[13]SPARQL 1.1 provides more intuitive negation, e.g. FILTER NOT EXISTS ?who rda:dateOfDeath ?ddate, although the Web NDLA does not support this functionality.

```
    ?uri xl:prefLabel [ ndl:transcription ?yomi ] ;
    #any graph pattern
    FILTER (lang(?yomi) = "ja-Latn")
}
```

2.6.4 Regular Expressions

The regular expression, introduced by `regex()` function, is a method to evaluate a string with flexible pattern. Partial matching to a string is also executed by a regular expression function.

The Web NDLA supports regular expression notations in Table 11[14].

Table 11: The Web NDLA Regular Expression

Notation	Functionality
`.`	matches any single character
`*`	matches zero or more times of the pattern immediately before
`+`	matches one or more times of the pattern immediately before
`?`	matches zero or one of the pattern immediately before
`^`	matches the head of a string
`$`	matches the tail of a string
`()`	grouping the enclosed patterns
`|`	choice of a pattern in a group (OR)
`{}`	specifies the number of repetition of the pattern immediately before
`[...]`	character class (set or range of characters)
`[^...]`	negation of the character class

The regular expression function is used in a form of `regex(text, pattern)`. For partial match, the pattern is the desired partial string.

For example, following query will find headings that contain "図書館".

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER regex(?label, "図書館")
}
```

The next query will find headings that consist of three capital letters.

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
```

[14]SPARQL Regular Expression follows the definition in XQuery/XPath specification, however, the Web NDLA does not support escape and meta characters with backslash at this moment.

```
    FILTER regex(?label, "^[A-Z]{3}$")
}
```

The `regex()` function may have a flag as the third argument that controls matching behavior. The Web NDLA supports `"i"` flag that makes the search case-insensitive. In the next query, `?label` will match either "internet", "Internet" or "INTERNET".

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER regex(?label, "internet", "i")
}
```

A `FILTER` can have multiple expressions combined by a logical operator. The next query will find headings that contain either "インターネット" or "Internet".

```
SELECT * WHERE {
    ?uri rdfs:label ?label .
    FILTER ( regex(?label, "インターネット") || regex(?label, "Internet") )
}
```

Note concatenation of two expressions by `&&` is equivalent to writing two `FILTER` clauses.

2.7 Solution Sequences and Modifiers

Sequence modifiers canbe used to make solutions an ordered sequence or to restrict numbers of solutions.

2.7.1 LIMIT and OFFSET

`LIMIT` clause after `WHERE` clause restricts the maximum number of solutions to be returned. `OFFSET` clause controls the starting solution to be returned in whole sequence.

The next example will return the first 10 headings.

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} LIMIT 10
```

The next example will return headings from 6 to 10 (`OFFSET` specifies how many solutions to be skipped from the top). The order of `LIMIT` and `OFFSET` clauses is not significant.

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} LIMIT 10 OFFSET 5
```

At this moment, the Web NDLA is configured to return at most 100 results. Therefore, the maximum number of results per request is 100, even if `LIMIT` clause tells more. Please use `OFFSET` clause in order to get results after 101.

```
SELECT * WHERE {
    ?uri rdfs:label ?label
} OFFSET 100
```

2.7.2 Sorting

`ORDER BY` keyword followed by space separated sort key variables establishes the order of the solution sequence. `DESC()` / `ASC()` modifiers indicate the enclosed variable is a descending / ascending key, respectively. Without enclosing modifier, a variable is treated as an ascending key.

```
SELECT * WHERE {
    ?uri rdfs:label ?label ;
        dct:modified ?moddate .
} ORDER BY ?moddate
```

Order modifiers can be combined with `LIMIT` and `OFFSET` clause to retrieve the sorted slice of the solutions. In this case, write `ORDER BY` first, then `LIMIT` and `OFFSET`.

2.7.3 Eliminating Duplications

In some graph patterns, it is possible that some solutions share the same set of bindings from variables to values. `DISTINCT` modifier after `SELECT` ensures that those duplications are eliminated from the solution set.

```
SELECT DISTINCT ?type WHERE {
    ?s a ?type .
}
```

`REDUCED` modifier also eliminates duplications, but does not guarantee uniqueness. While both modifiers generate the same solution set in most cases, `REDUCED` would return the results faster for large data sets, since the heavy computational burden required by complete elimination would be obviated.

2.7.4 Aggregates

Aggregates such as grouping the solutions and counting the number of results are introduced in SPARQL 1.1. Although the Web NDLA (ARC2 library) implements SPARQL 1.0, there are some aggregate functions available.

In order to count the number of the results, use set function `COUNT()` with the target variable as its argument, and assign the counted number to a new variable by `AS` keyword, in `SELECT` clause.

For example, the next query will find the number of authority records of people who were born in 1960, in the Web NDLA.

```
SELECT (COUNT(?who) AS ?howmany) WHERE {
    ?who rda:dateOfBirth "1960" .
}
```

GROUP BY keyword followed by variables, after WHERE clause, divides results into groups. Then the aggregate value is calculated for each group. The next example will find the year-by-year number of authority records of people who were born in 1900 or later.

```
SELECT ?byear (COUNT(?who) AS ?howmany) WHERE {
    ?who rda:dateOfBirth ?byear .
    FILTER (?byear >= 1900)
} GROUP BY ?byear
ORDER BY ?byear
```

Set functions for aggregates are MAX(), MIN(), AVG() and SUM() as well as COUNT() (AVG() and SUM() are applicable only for numerical values). The next example will find the latest and the oldest year of birth in authority records of personal name.

```
SELECT (MIN(?byear) AS ?past) (MAX(?byear) AS ?recent) WHERE {
    ?who rdf:dateOfBirth ?byear .
}
```

2.8 Query Forms and Results

SPARQL has four query forms: SELECT, ASK, CONSTRUCT and DESCRIBE.

2.8.1 SELECT

To find the values that match the pattern, use SELECT query.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?subj2 ?label
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

With the above SELECT query, the Web NDLA will return the following results (for XML format).

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="subj2"/>
    <variable name="label"/>
  </head>
  <results>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00969901</uri>
      </binding>
      <binding name="label">
        <literal>バーチャルプライベートネットワーク</literal>
      </binding>
    </result>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/00865280</uri>
      </binding>
      <binding name="label">
        <literal>イントラネット</literal>
      </binding>
    </result>
    <result>
      <binding name="subj2">
        <uri>http://id.ndl.go.jp/auth/ndlsh/01017771</uri>
      </binding>
      <binding name="label">
        <literal>セマンティックウェブ</literal>
      </binding>
    </result>
  </results>
</sparql>
```

The `<head>` element enumerates the variable names in the results set, and the `<results>` element contains `<result>` elements for each matching variable set. See 3.1 for the detail of the result format.

2.8.2 ASK

To determine whether there are any partial graphs that match the pattern (without retrieving the values), use `ASK` query. Because this query ignores variable values, the keyword is immediately followed by the `WHERE` clause.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
ASK WHERE {
```

```
    ?subj1 rdfs:label "インターネット"
}
```

With the above `ASK` query, the Web NDLA will return the following results (for XML format).

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head></head>
  <boolean>true</boolean>
</sparql>
```

If no matching partial graph is found, the content of `<boolean>` is `false`.

2.8.3 CONSTRUCT

The matching variable values can be used to construct another RDF graph. Write the pattern of the new graph with similar syntax, put it in `{}` after the keyword `CONSTRUCT`, then place the `WHERE` clause.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX ex: <http://example.org/terms#>
CONSTRUCT {
    ?subj1 ex:下位語 ?label .          # 下位語 means narrower term
}
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

With the above `CONSTRUCT` query, the Web NDLA will return the following graph (for Turtle format).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns0: <http://example.org/terms#> .

<http://id.ndl.go.jp/auth/ndlsh/00841024>
    ns0:下位語 "バーチャルプライベートネットワーク" ,  # virtual private network
        "イントラネット" ,                              # Intranet
        "セマンティックウェブ" .                        # Semantic Web
```

Note the prefixes in the returned Turtle might be different from those specified in the query.
If `output` parameter is `xml`, the graph is returned in RDF/XML format.

2.8.4 DESCRIBE

With `DESCRIBE` query, a user can retrieve an RDF graph which is about the resources (variable values) that match the condition. In the Web NDLA, this will be a "description graph" that consists of triples whose subjects are the value URIs, plus triples connected to them via blank nodes.

```
DESCRIBE <http://id.ndl.go.jp/auth/ndlsh/00841024>
```

With the above `DESCRIBE` query, the Web NDLA will return the same RDF graph as to be retrieved from the URI with suffix `.ttl` or `.rdf`.

A description graph can be obtained for the resources that match a graph pattern in `WHERE` clause. If there are multiple matches or multiple variables are specified, the merge of the "description graphs" will be returned.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
DESCRIBE ?subj2
WHERE {
    ?subj1 rdfs:label "インターネット" ;
        skos:narrower ?subj2 .
    ?subj2 rdfs:label ?label .
}
```

The "description graph" will be empty if the value of the variable is a literal.

3 API Parameters and Result Formats

An API request is to be sent to the following URI, with URL encoded query string as `query` parameter value and result format as `output` parameter value, as shown in 1.1.2.

```
http://id.ndl.go.jp/auth/ndla
```

The variables binding list and TRUTH value, as the result of `SELECT` and `ASK` query respectively, will be returned with XML or JSON result format according to `output` parameter value.

3.1 XML Result Format

When the `output` parameter value is `xml`, the result set is returned in SPARQL Query Results XML Format[15].

This format is, as shown in the example in 2.8.1, an XML document in namespace `http://www.w3.org/2005/sparql` whose root `sparql` element has `head` and `results` (`boolean` for ASK query) elements.

The `head` element has `variable` elements in query order, where `name` attribute is the name of each variable. The `head` element is empty for `ASK` query.

```
<head>
  <variable name="subj2"/>
  <variable name="label"/>
</head>
```

The `results` element contains zero or more `result` elements which have `binding` elements for each bound variable. A `binding` element has a `name` attribute corresponding to the name of variable, and has one child element `<uri>`, `<literal>` or `<bnode>` (depending on the value type) whose content is the value of the variable.

If the value is literal and has language tag, `<literal>` has `xml:lang` attribute. If it is a typed literal, the element has `datatype` attribute whose value is the datatype URI.

```
<results>
  <result>
    <binding name="subj2">
      <uri>http://id.ndl.go.jp/auth/ndlsh/00969901</uri>
    </binding>
    <binding name="label">
      <literal>バーチャルプライベートネットワーク</literal>
    </binding>
  </result>
  ...
</results>
```

[15] http://www.w3.org/TR/rdf-sparql-XMLres/

For `ASK` query, there is a `<boolean>` element instead of `<results>`, whose content is `true` or `false`, as shown in the example in 2.8.2.

3.2 JSON Result Format

When the `output` parameter value is `json`, the result set is returned in SPARQL 1.1 Query Results JSON Format[16].

This is a JSON document whose top most object has `head` and `results` (`boolean` for `ASK` query) property. The result in JSON format for the query in 2.8.1 will be as follows.

```
{
  "head": {
    "vars": [
      "subj2",
      "label"
    ]
  },
  "results": {
    "bindings": [
      {
        "subj2": {
          "type": "uri",
          "value": "http://id.ndl.go.jp/auth/ndlsh/00969901"
        },
        "label": {
          "type": "literal",
          "value": "\u30d0\u30fc\u30c1\u30e3\u30eb\u30d7\u30e9..."
        }
      },
      ....
    ]
  }
}
```

The `head` property value is an object which has a `vars` property, whose value is an array of variable names.

The `results` property is an object which has a `bindings` property, whose value is an array of objects of result sets, where bound variable names are properties. Each property value is an object, which has a `type` property to tell whether the value is URI, blank node or literal, and a `value` property to show the value.

Note that `value` property value will be escaped as `\u` + Unicode number for non-ASCII characters, as shown in the above example.

[16] http://www.w3.org/TR/sparql11-results-json/

4 RDF Graph of the Authority Records and Applied Examples

This chapter explains the model (graph structure) of the authority records in the Web NDLA, and presents some examples to apply SPARQL constructs discussed in the previous chapters against the graph.

4.1 Authority Records of Personal Name, Family Name and Corporate Name

The authority record resource is distinguished from the entity resource that corresponds to the real world thing (person etc.) in the graph of personal name, family name and corporate name.

An authority record has such properties as preferred label, alternative label, source and related link, to name the few. Preferred label and alternative label have structures via blank nodes in order to provide literal forms and yomi (transcriptions) together.

An entity resource describes such real world attributes as birth year of a person, establish year or history of a corporate. An authority record resource and its corresponding entity resource are related by `foaf:primaryTopic` (Figure 4).

4.1.1 To Find Birth Year, Preferred Label and Kana Transcription of a Person

Provided that a person's name is known, compose a `SELECT` query where the name is the object of `foaf:name` and other values in question are variables.

```
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xl: <http://www.w3.org/2008/05/skos-xl#>
PREFIX ndl: <http://ndl.go.jp/dcndl/terms/>
SELECT * WHERE {
    ?auth
        foaf:primaryTopic ?entity ;
        xl:prefLabel [
            xl:literalForm ?preflabel ;
            ndl:transcription ?yomi ] .
    ?entity
        rda:dateOfBirth ?birth ;
        rda:dateOfDeath ?death ;
        foaf:name "夏目漱石".            # Natsume, Soseki
    FILTER (lang(?yomi) = "ja-Kana")
}
```

Note that preferred label and `rdfs:label` value are normalized in the form of family name, first name and birth-death year, e.g. "夏目, 漱石, 1867-1916". The above example uses `foaf:name` of the entity resource to search with a common name format such as concatenated name (for Japanese) or first name then last name form.

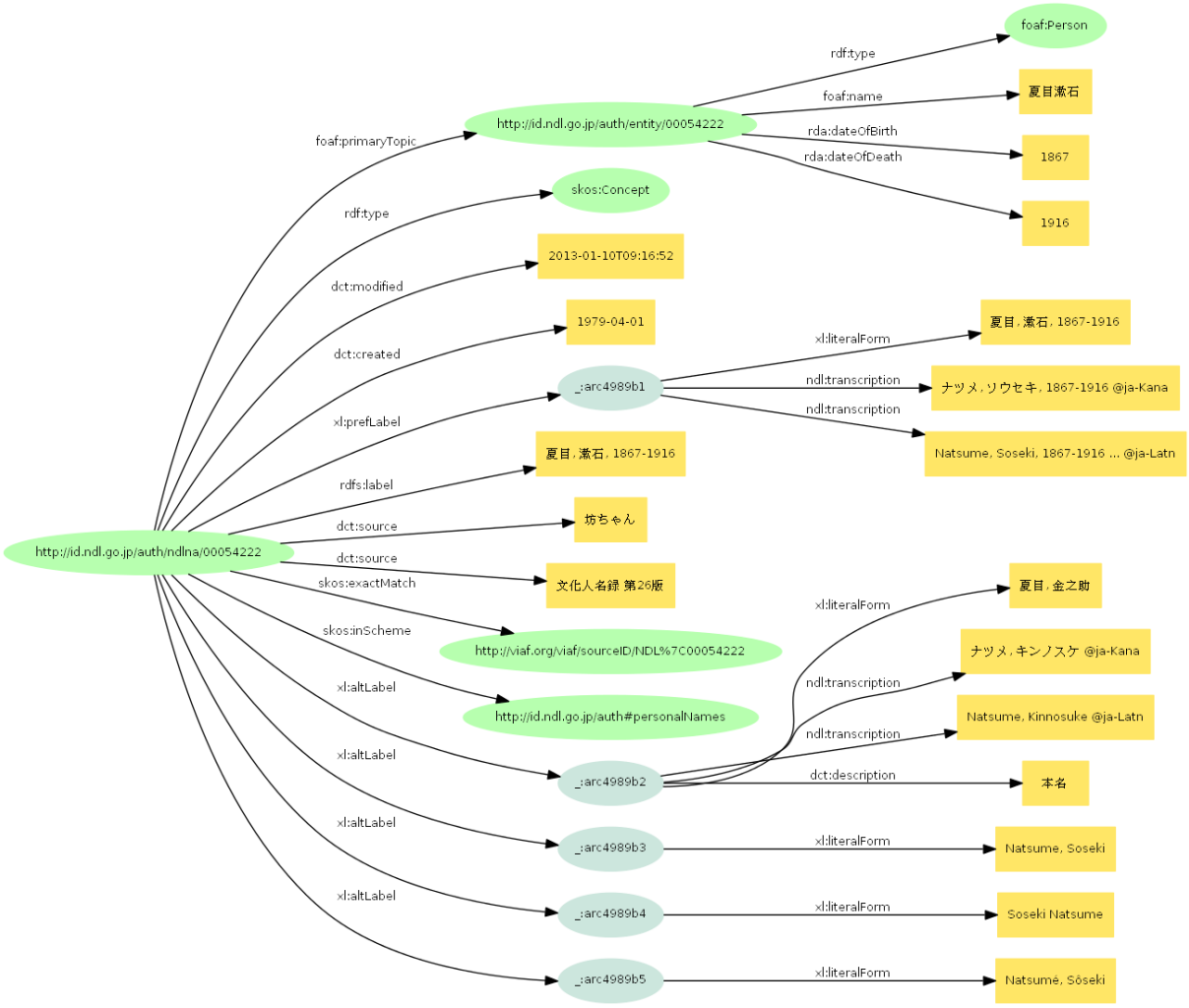

Figure 4: Name authority graph of "夏目漱石". A node .../ndlna/00054222 in the left of the figure represents the authority record, and .../entity/00054222 at the top represents the entity resource.

Since the preferred label has both Kana and Romaji yomi (`ndl:transcription`), use `FILTER` to restrict the language tag.

4.1.2 To Search Person Born in 11th Century, Find VIAF Link and Sort by Birth Year

To find person born in 11th century, express the condition "born between 1001 and 1100" with the logical operator in the `FILTER` clause. An authority record links to VIAF with `skos:exactMatch`. Variable `?birth` is used to sort the result in addition to `FILTER` expressions.

```
PREFIX rda: <http://RDVocab.info/ElementsGr2/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT * WHERE {
    ?auth
        foaf:primaryTopic [
            rda:dateOfBirth ?birth ;
            foaf:name ?name ] ;
        skos:exactMatch ?viaf .
    FILTER (?birth >= 1001 && ?birth <= 1100)
} ORDER BY ?birth
```

Note that the Web NDLA does not allow more than 100 results to be retrieved for a single request. Use `OFFSET` clause repeatedly to get more than 100 results (See 2.7.1).

4.2 Authority Records of Geographical Name, Uniform Title, Subject Heading and Subject Sub-Division

These authority records do not have corresponding entity resources, hence they have flat structure graphs (though preferred label and alternative label have their substructures to provide literal form and transcriptions together). Subject headings have more properties than name authorities in order to describe broader, narrower, related terms or classifications (Figure 5).

4.2.1 To Find Broader or Narrower Terms of Subject Heading

Since subject headings are not normalized as name authorities, simply use `rdfs:label` as the property of the known heading, and let broader or narrower terms be variables. The next example will find the broader term of "インターネット".

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        rdfs:label "インターネット" ;
        skos:broader ?broader .
```

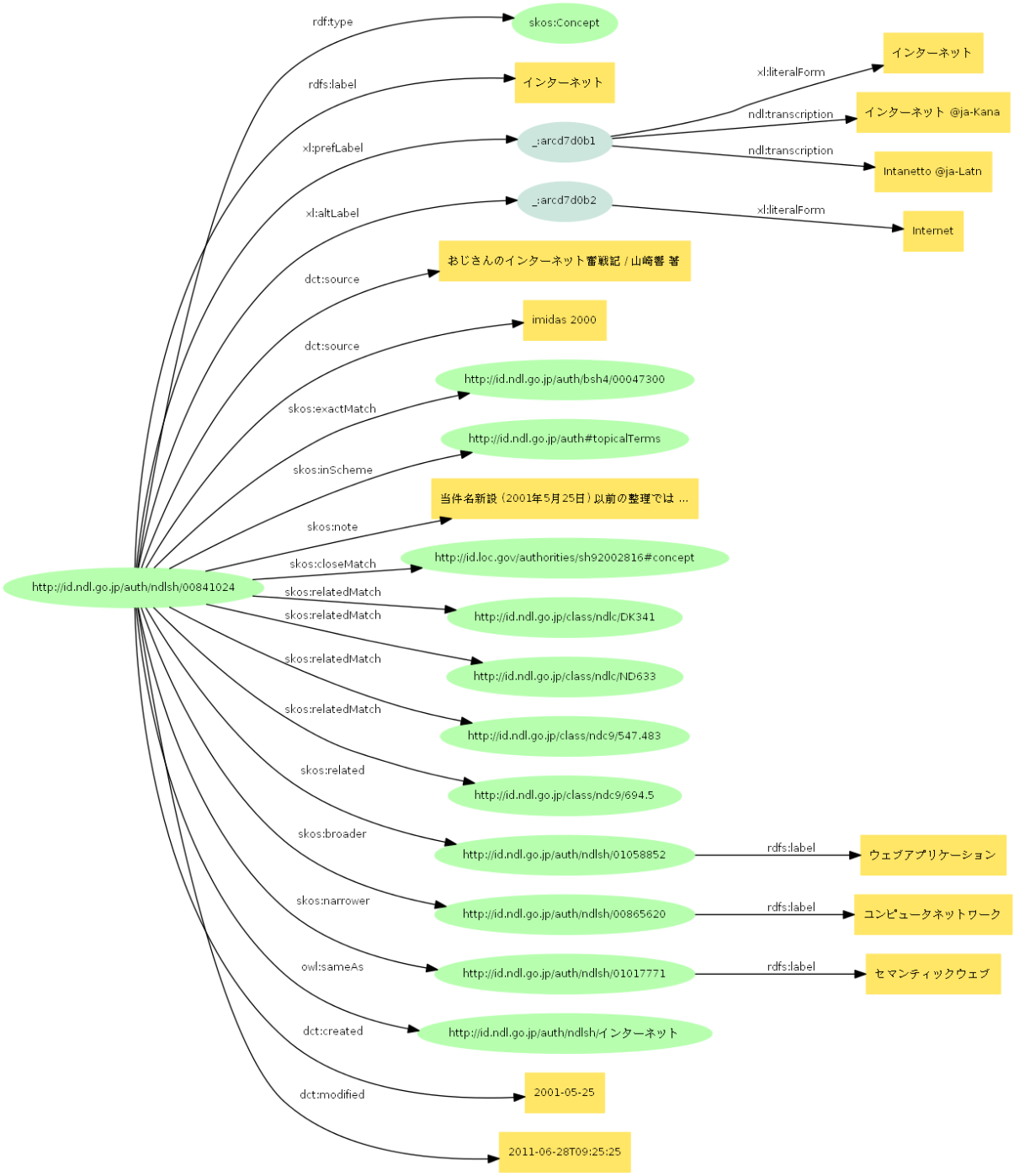

Figure 5: RDF graph of subject heading "インターネット". Some narrower and related terms are omitted for simplicity.

```
    ?broader rdfs:label ?label .
}
```

4.2.2 To Find Subject Headings with Some Classifications

In order to find subject headings that have classification of NDLC "DM225" which is identified by a URI, construct a query with that URI as the object of `skos:relatedMatch`.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DM225> ;
        rdfs:label ?label .
}
```

To find subject headings classified as both NDLC "DK341" and NDC9 "694.5", let the graph pattern have two objects of `skos:relatedMatch` (NDC9 is also identified by a URI, with different base URI).

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
    ?subj
        skos:relatedMatch <http://id.ndl.go.jp/class/ndlc/DK341> ,
            <http://id.ndl.go.jp/class/ndc9/694.5> ;
        rdfs:label ?label .
}
```